

An Object-Based Approach to Medical Process Automation

Dipayan Gangopadhyay, Ph.D. and Peter Y.F. Wu, Ph.D.

• IBM Thomas J. Watson Research Center

P.O. Box 704, Yorktown Heights, NY 10598

e-mail: dipayan@watson.ibm.com, pwu@watson.ibm.com

The medical events of providers rendering services for patient care are necessarily interrelated. A clinical information system must reliably record these events and relate the information about their inter-dependency. The quality of clinical information therefore depends crucially on the proper coordination and tracking of these events according to established protocols. We introduce an object-based approach to define medical processes for their automation. For each medical process, we capture in one logical unit, an event-driven mechanism to coordinate of inter-dependent medical events of the process, and the data relevant to the process. We call the encapsulated unit a form object. Moreover, the form object may contain sub-objects each of which also encapsulates its own protocol knowledge and relevant information. In contrast to other approaches, the form object facilitates for administering medical processes due to its locality of definition, and its event-driven paradigm reflects medical events more naturally.

1. INTRODUCTION

Consider the process of ordering medication for a patient. An advising nurse may initiate the order, but it requires the doctor's approval before the pharmacy can fill the prescription. When the pharmacy fills the prescription, the action must also result in a record for the drugs dispensed, and the completion of the medication order entered into the patient's medical record. In short, medical processes involve *multiple parties* generating *asynchronous events* over time (such as "doctor has approved" and "prescription filled"). Most importantly, the results must be *recorded reliably*

in the patients' medical records. A clinical information system (CIS) should coordinate and track these concurrent events generated by these spatially distributed parties. In fact, accurate, reliable and timely execution of medical processes is essential to the overall accuracy of the patients' medical records.

Moreover, different health care organizations may have different needs and specific requirements according to their policies and regulatory mandates. Their protocols for medical processes may have to be somewhat different. Even within the same organization, protocols may need to change over time. For example, using results from time and motion studies, an organization may modify the protocols to improve efficiency and robustness. To adapt to organizational differences and to accommodate changes in protocols, the CIS must support an architecture which drives the execution of the medical processes from definition, rather than hard-coding the automated protocols in the constituent software programs. This calls for a new level of flexibility in the structure of the system, which separates the definition of protocols from the automated execution of the processes according to their defined protocols.

We present an object-based approach to automate medical processes. In our approach, we capture in one unit, the coordination of medical events and the related data together. Such a unit is an object in the system, which we call a **form object**. We gather data as attributes in the form object, and we specify its behavior in a finite state machine, an event-driven mechanism to coordinate the events in the medical process. Furthermore,

a form object, in turn, can contain other sub-objects, each of which also encapsulates its own protocol knowledge and relevant data items, as well as its sub-objects. The reader may refer to [1] for a more complete description of the object modeling method and notations.

Our approach defines as one encapsulated unit, all relevant information items and protocol knowledge for a medical process. Such locality of definition serves as the key to manage and facilitate changes in the protocols. More, the hierarchical decomposition of objects into sub-objects (instead of process decomposition) is our means to manage complexity, without loss to the locality of definition. Such decomposition enables our approach to scale up to complex medical protocols without making the finite state machine of any object too large and cumbersome. Besides, since the form objects encapsulate protocol knowledge along with related data, and we can specify them precisely, we can drive the execution of medical processes from their definition. Therefore, we can easily modify the definition of a form object to change the protocol, consequently resulting in changes in the execution of the medical process. The separation of definition from execution enables us to add new protocols and change existing ones without making an impact on the run-time system. Limited by space, this paper will concentrate on medical process definition and will not discuss further on the execution environment.

We currently have a prototype system running on a Unix workstation with X-windows and Motif. The prototype system is a definition tool to visually create the form objects with interactive graphics, and a simulated execution environment to verify the processes defined the form objects. The reader may refer to [2] for more information about the system.

Some earlier studies concentrated on protecting data integrity in clinical information systems [3,4,5]; there was also use of finite state machine to capture process knowledge [6]. Our approach distinguishes from others in its definition paradigm for medical processes: our approach provides locality of definition via encapsulation of relevant data and event-driven ordering of process steps in objects, and we support hierarchical de-

composition of these objects to manage complexity. What is germane here is not the use of finite state machines, but the encapsulation of process and data as objects and management of complexity by hierarchical decomposition of objects. We believe that our approach facilitates management and modification of protocols much easier than the traditional process decomposition approach where data and processes are not localized.

Section 2 will present our object-based approach to modeling the medical process. Section 3 gets into further details of medical process definition. We explain the attributes and the sub-objects in the data part of a form object, and the state diagram of a finite state machine which captures the medical process as the life-cycle of the data relevant to the process. Our explanation will take a running example of ordering new medication for illustration. Section 4 will discuss the advantages of our approach in comparison to some other approaches.

2. THE OBJECT-BASED APPROACH

We model the medical process in an object-based approach. In the CIS context, a medical process comprises a collection of information items, such as forms, and a coordinated sequence of operations by agents, such as health care providers, on the collected information. The central idea of our approach is to encapsulate data and the coordination of processing operations in an object, which we call the **form object**. Moreover, the form object may also contain sub-objects, each of which, in turn, encapsulates its own data items and process knowledge, as well as its own sub-objects. The hierarchical decomposition can therefore hide complexities at the appropriate levels of abstraction. Basically, the form object which models the medical process consists of two parts: data and behavior. The data part consists of the attributes of the form object, and the sub-objects it may contain. The behavior part is a finite state machine which specifies how the form object should respond to events which may happen. Under our model, the form object interacts with various agent objects in the CIS, as the medical process proceeds.

Consider the example of ordering new medication

for a patient. Let us call the process NewMedication. The NewMedication form object consists of, in the data type part, a sub-object Prescription and attributes about an Agent, the health care provider who initiates the NewMedication process, and the name of a doctor whose approval is required. In the behavior part, the NewMedication form object consists of a finite state machine to execute the NewMedication process. When a health care provider issues a new medication, he/she creates an instance of the NewMedication form object, filling in all the necessary information to initiate the process. The form object follows its behavior description in the state diagram to interact with other agent objects, as the NewMedication process proceeds. The state diagram is, in fact, the life-cycle of the data items in the process. Figure 1 depicts the form object interacting with another agent object in the execution of the medical process.

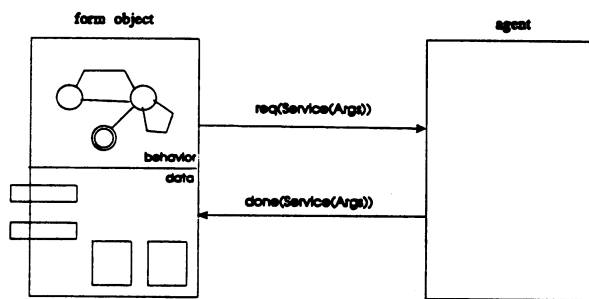


Figure 1. A form object interacting with another object

3. MEDICAL PROCESS DEFINITION

The form object defines the medical process. The form object declares the information items needed for processing, and describes the sequencing control to process the information. In our model, the behavior description is a state diagram of the finite state machine to execute the process, depicting the life-cycle of the data in the process. In this section, we will further explain the form object. The description here is an adaptation of the ObjChart notations [1,2] for concurrent objects and their behavior. Figure 2 illustrates a sample NewMedication form object.

The form object has two parts: data and behavior. The data part contains a prescription sub-object, and the attribute information concerning the agent who initiated the NewMedication process, and the doctor whose signature is needed to fill the prescription. The behavior part is a finite state machine, depicted by the state diagram, which specifies the protocol the NewMedication process must go through. Each process may have to go through certain number of states from start to finish. We call the initial state the **start** state, and the states at which the process will terminate the **accept** states. In our state diagram, ovals represent the states. We mark the accept states in double ovals, and we label each state of the process inside the ovals.

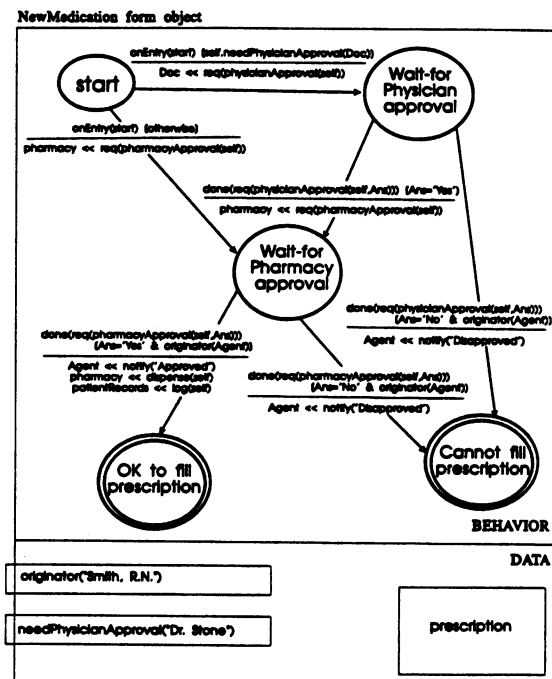


Figure 2. The NewMedication form object

From one state to another, the process goes through a state transition. Arcs connecting one oval to another represent the state transitions. Events trigger state transitions, when certain specified conditions are met. When state transition takes place, a list of associated actions will fire asynchronously. These actions usually involve interaction with other objects in the CIS to carry out the processing steps, which sometimes may results in other events. In the state diagram, we

label each arc in the form

Event[Conditions]
Actions

In the following, we will explain **Event**, **Conditions** and **Actions**, taking the NewMedication form object as an example.

3.1 Event

At any state (except the **accept** states), the finite state machine is waiting for some events which may trigger a state transition. An event usually signifies the receipt of a message. Our example illustrates two kinds of events:

1. **onEntry(S)**: This event happens whenever the state machine enters state **S**;
2. **done(Service(*Args*))**: This event happens when an agent sends a message to this form object to notify completion of service requested.

3.2 Conditions

A condition is a boolean expression of simple relational comparisons or query predicates. A simple relational comparison has the form such as $X < Y$, where X and Y are variables instantiated in the return arguments of the associated event, or literal constants. A query predicate has the form **obj.prop(*Args*)** where **obj** refers to some object contained in the data part of the form object, so that we can synchronously evaluate the expression. The expression **obj.prop(*Args*)** evaluates to **true** if **obj** has the property **prop(*Args*)** for some bindings of the arguments (*Args*). To refer to **self**, the form object itself, we can omit **obj**. A special symbol **otherwise** represents the condition when none of the other conditions with the same event from the same state evaluates to **true**.

In our NewMedication example, at the **start** state, the **onEntry(start)** event may trigger a state transition. If the form object has the property so that "**needPhysicianApproval(*Doc*)**" evaluates to **true** with the argument *Doc* instantiated to some physician, the state transition takes place and the associated actions fire concomitantly.

3.3 Actions

Associated with each arc is a list of actions. Each

action has the form

T \ll **message(*Args*)**

where **T** is an Agent object in the CIS, representing a worker or managing some resources. The action denotes sending an asynchronous message "**message(*Args*)**" to **T**. In our model, the action usually involves sending a service request, a message of the form "**req(Service(*Args*))**" to some Agent object.

In the NewMedication example, when state transition goes from **start** state to **Wait-for-physician-approval** state, the message "**req(physicianApproval(self))**" is sent to the **Doc** identified on the form object as the doctor needed to approve this.

The NewMedication form object is then in the state **Wait-for-Physician-Approval**. The event "**done(req(physicianApproval(self,*Ans*)))**" may trigger another state transition, whenever the doctor responds to the request to approve or disapprove the prescription. Depending on the result in *Ans*, the form object moves on to other states until it comes to an **accept** state to finish its life-cycle. In its behavior part, the state diagram describes the proper sequencing to execute the process.

4. DISCUSSION

We have described an object-based approach to define a medical process as the life-cycle of information items affected by the process. An event-driven paradigm captures such a life-cycle in a finite state machine where completion of service rendered by service providers will report the "**done**" events to the corresponding form object.

We have adapted the visual environment from an object modeling tool, ObjChart Builder [2] to define these objects and their state machine driven behavior graphically. We define medical processes in form objects using the tool. In the run-time environment, distributed fail-safe servers in the networked CIS relay events to the form objects as the medical processes proceed. During the execution, the form objects communicate with other agent objects, which reside on workstations serving the health care providers.

We believe that our object-based definition approach has several advantages. First, object-based approach encapsulates procedural steps and the relevant data as one integral unit. The encapsulated locality facilitates modification of the definition, should the protocol need re-definition to adapt to organizational or policy changes, or the like. In contrast, a non-object-based approach often must decompose the process into a sequence of smaller steps, separate from the data that each of the steps operates on. The separation does not preserve the locality of definition and changes become harder to administer. Second, by choosing an event-driven paradigm, our approach can express naturally the various exceptional conditions which very often occur in real-life situations. Each new exceptional condition can be handled by adding a new arc, for example, to the state machine. This is in contrast to data-flow approaches where the process definition describes how the data object will flow from agent to agent; with such data-flow approach, each exceptional condition makes the flow more complicated although the normal flow may be simple. Finally, our framework of concurrent objects, communicating via asynchronous messages, is natural for capturing the inherent concurrency present in the real world. For example, a doctor goes on to serve the next patient after issuing a prescription while the pharmacy may proceed on to fill the prescription, dispensing the medicine to the patient.

An alternative approach to define medical processes is to model them as interaction protocols among a group of providers [7]. For example, we can describe the process of prescribing new medication in a sequence of message exchanges among the providers, namely the originator, the co-signee physician, and the dispensing pharmacist. It turns out that the approach to encapsulate finite state machine and data together can also be applied to model inter-provider protocols as objects, with the state machines to capture the protocol and the sub-objects for the relevant data need to be recorded across steps of the protocol.

Reference

- [1] D. Gangopadhyay and S. Mitra. *ObjChart: Tangible Specification of Concurrent Object Behavior*. Proceedings of ECOOP'93 - 7th European Conference on Object-Oriented Programming, published in Lecture Notes in Computer Science 707, Springer-Verlag, July 1993, pp.432-457.
- [2] D. Gangopadhyay, S. Mitra and S.S. Dhaliwal. *ObjChart Builder: An Environment for Executing Visual Object Models*. To be presented in TOOLS-USA'93, 11th International Conference on Technology of Object-Oriented Languages and Systems, Santa Barbara, California, August 1993.
- [3] M.S. Roberts, E.M. Dreese, N. Hurley, N. Zullo, and M. Peterson. *Blending Administrative and Clinical Needs: The Development of a Referring Physician Database and Automatic Referral Letter*. Proceedings of the 15th Annual Symposium on Computer Applications in Medical Care, McGraw-Hill, 1991, pp.559-563.
- [4] S. Johnson, C Friedman, J.J. Cimino, T. Clark, G. Hripcsak, and P.D. Clayton. *Conceptual Data Model for a Central Patient Database*. Proceedings of the 15th Annual Symposium on Computer Applications in Medical Care, McGraw-Hill, 1991, pp.381-385.
- [5] A.J. Chandrasekhar, and R.N. Price. *A Protocol System to Satisfy Research Specific, Disease Specific, Physician Specific, and Patient Specific Informational Needs*. Proceedings of the 16th Annual Symposium on Computer Applications in Medical Care, McGraw-Hill, 1992, pp.724-728.
- [6] S.W. Tu, M.G. Kahn, M.A. Musen, J.C. Ferguson, E.H. Shortliffe, and L.M. Fagan. *Episodic Skeletal-Plan Refinement Based on Temporal Data*. Communications of ACM, Vol.32, No.12, December 1989, pp.1439-1455.
- [7] A.L. Scherr. *A New Approach to Business Processes*. IBM Systems Journal, Vol.32, No.1, 1993, pp.80-98.